Lookahead Scheduling Requests for Efficient Paging (Algorithms and Theory of Computing)

AUTHOR(S):

Kiniwa, Jun; Kameda, Tiko[Tsunehiko]

数理解析研究所講究録
1041 巻 1998 年 27-34

Lookahead Scheduling Requests for Efficient Paging

Jun Kiniwa(木庭 淳)
Dept. of Management Science,
Kobe Univ. of Commerce, Japan

Tiko Kameda(亀田恒彦)
School of Computing Science,
Simon Fraser Univ., Canada

February 2, 1998

Abstract

This paper studies the effect of scheduling (reordering) page requests on the page hit ratio. First, we define the residence interval of a page, the time duration during which the page resides in the cache continuously, for off-line page request sequences. Using the residence interval, we show how to reorder requests to reduce the number of page misses. In particular, we show two page replacement policies are useful to minimize some criteria. Second, we consider the "semi-online" model, which assumes that some future requests are known. This model is applicable in practice if unserviced requests are kept in a queue and reordered before execution. Then we develop an efficient semi-online algorithm which selects a page to be evicted from k candidates, and analyze appropriate k. Third, the performance of our scheduling method is examined by simulation, by varying the mean inter-arrival time of requests and cache size. The results indicate that our method outperforms the popular *LRU* (Least Recently Used) and "extended *LRU*."

1 Introduction

Caching is a standard technique to achieve an effective speed up of memory access by providing a small amount of fast (expensive) cache memory. A 2-level hierarchical memory, consisting normally of a main memory cache and a disk storage, is currently most common. Our model reflects such a memory system with a cache which can contain m pages and a slow memory which can contain a large number of pages. Our model also has an input queue which keeps unserviced page requests.

If the requested page is in the cache, i.e., a page *hit* occurs, then the access incurs little cost (in terms of response time). If the requested page is not in the cache, i.e., a page *miss* or *fault* occurs, then the access is much more expensive. In case of a page miss, we have to flush some page out (if it's *dirty*, i.e., it is different from its image in the slow memory) and fetch the requested one from the slow memory. A page replacement policy determines which page to purge from the cache upon a page miss. Usual page replacement policies do not take future requests into account, while our model can examine a queue of unserviced requests to make the optimal decision using this information. In this paper, for simplicity, we will refer to a page hit (page miss), simply as a hit (miss).

We present a new method for preprocessing the input request sequence to reduce the number of misses. Intuitively, we can reduce the number of misses, if we are allowed to reorder input requests. The main contributions of this paper are:

1. A set of new operations which improve the hit ratio for off-line scheduling.
2. They are also effective for "semi-online" scheduling.

3. Performance is evaluated by simulation, comparing with LRU and "extended LRU."

First, we define the *residence interval* of a page, which is the time duration during which the page resides in the cache continuously. If a page is evicted and called back again in processing a sequence of requests, we may be able to extend the page's residence interval slightly and reduce the number of misses. That is, if we are allowed to reorder the requests in advance, we can get a history with the minimum number of misses. Second, we observe that the operation above can be used even if all the future requests are not known. This will be the case in practice when the arrived requests are kept in the input queue, which we call the "semi-online" scheduling, to contrast it with the on-line scheduling, where no future requests are known. We develop a semi-online algorithm which selects a page to be evicted from k candidates. What value is appropriate for k is also analyzed in terms of two criteria. Third, we investigate the performance of our method by simulation, assuming that the queue length is determined by the following parameters.

- mean inter-arrival time of requests
- hit ratio and the time overhead associated with a hit and a miss

Our simulation results show that our method outperforms the conventional LRU and "extended LRU," which we call "partial LRU."

Our work reported here is a continuation of the study on page replacement algorithms, started by Belady [3]. He presented an optimal off-line algorithm *MIN*. Since then, a large number of papers on the problem has been published on the subject [9, 10], and various algorithms, e.g., *LRU*, *FIFO* and *MFU*, have been proposed. It is of interest to evaluate different page replacement algorithms. Sleator and Tarjan[8] studied a well-known evaluation method, called *competitive analysis*. It has been widely applied to on-line algorithms, because they are difficult to analyze by conventional methods. This method, however, is sometimes criticized to be too pessimistic. Recently, [1, 4, 5, 7, 11] investigated the effect of a lookahead policy on the paging problem. The weak model[1] of lookahead policy has a queue of length l, containing l future requests. However, Ben-David and Borodin [4], and also Koutsoupias and Papadimitriou [7] showed that the weak model of lookahead does not improve the competitive ratio, which indicates the limit of lookahead policies. Albers [1] and Breslauer [5] showed that other models of lookahead, e.g., containing at most l miss requests in the queue, can improve the competitive ratio. Yanbe and Sakurai [11] discovered that game theory can evaluate the performance of lookahead policy under the weak model. Previously, some authors studied the effect of scheduling requests for disk scanning [6].

The rest of this paper is organized as follows. Section 2 investigates off-line scheduling. We develop an operation to improve the hit ratio, and show that our algorithm is effective. Section 3 applies it to semi-online scheduling, assuming that we know some future requests. We compare our algorithm with the conventional *LRU* and "extended *LRU*" by simulation. Section 4 concludes the paper.

2 Off-line Scheduling

2.1 Definitions

To begin with, we give some definitions.

Definition 2.1 *Let $U = \{a, b, \ldots\}$ be the set of all pages. A* **history** *of length n is an onto mapping $h : \{1, 2, \ldots, n\} \to U(h)$, where $U(h) \subseteq U$. If the ith request of h is for $q \in U$, i.e., $h(i) = q$, then we refer to this request as $q^{(i)}$. The* **distance** *from $p^{(i)}$ to $q^{(j)}$ in h is defined by $d(p^{(i)}, q^{(j)}) = |i - j|$. If the index is clear from the context, we sometimes use q instead of $q^{(i)}$. For technical reasons, we assume that a history h always starts with the* **initial sequence** *I_0, where every request in $U(h)$ appears exactly one. A cache is a set of m pages, called a cache size. If replacement policy $\mathcal{R}$ is used*

[1] Albers[1] named it in contrast to her "strong model".

in processing request sequence represented by history h, $C_{\mathcal{R}}(q^{(i)};h)$ denotes the contents of the cache at the time $q^{(i)} \in U(h)$ is requested. We may omit some of the subscripts/superscripts/arguments from $C_{\mathcal{R}}(q^{(i)};h)$, when they are clear from the context. ∎

To motivate our idea, let us consider a simple example. Suppose that the cache currently holds three pages, $\{a,b,c\}$, and that the current request is for d. Suppose also that pages in the cache will be requested in the order $\cdots a \cdots b \cdots c \cdots$, where the replacement policy $\mathcal{R}$ evicts page c now. Then we know that the next request for c will cause a miss. We shift the request for c to immediately before the request for d, resulting in the shifted one does not cause a miss. If we similarly shift all the future requests for c, the page c can be evicted without causing future page miss. Note that some pages in $C_{\mathcal{R}}(q^{(i)};h)$, e.g., the page c in the example above, may not be requested by h in the future.

The **residence interval** of a page a is the maximal time interval during which a is in the cache. For any two requests x and y in a history h, let $[x,y]$ denote the time interval from when x is requested until when y is requested. We have the following important proposition.

Proposition 2.1 *For an arbitrary request a, if there are two miss requests $a^{(i)}, a^{(k)}$ such that there is no miss request $a^{(l)}$ with $i < l < k$ in the history h, then there is a miss request $b^{(j)}$ $(i < j < k)$ which evicts the page a, and the interval $[a^{(i)}, b^{(j)}]$ is a subset of a residence interval of a.*

Proof In the interval $[a^{(i)}, a^{(k)}]$, page a must be evicted once. Let $b^{(j)}$ $(i < j < k)$ be the request that causes the eviction. Then there must exist a miss request $b^{(j)}$ which evicts the page a such that $[a^{(i)}, b^{(j)}]$ is a subset of a residence interval of a. ∎

2.2 Algorithm

We now present an off-line optimal algorithm **SRRI** (Shifting Requests to Residence Interval). It converts any history to one causing the minimum number of misses. Further, we can reduce the number of reordering requests or maximum delay if we choose appropriate replacement policy. Hence we consider *LSD* (Least Sum of Distances) and *LFU* (Least Frequently Used) here as replacement policy $\mathcal{R}$, because the former minimizes the number of reordering and guarantees no starvation, and the latter minimizes the maximum delay. The *LSD* counts the sum of distances from the current request for each request of page in the cache, and picks the least one. The *LFU* counts the number of requests for each page in the cache, and picks the least one. Suppose that $a^{(i)}$ causes an eviction of page x under the replacement policy $\mathcal{R}$. Let $\sigma_{\mathcal{R}}(a^{(i)})$ be the request for x such that there is no $x^{(j)}$ $(i < j)$ in the interval $[a^{(i)}, \sigma_{\mathcal{R}}(a^{(i)})]$.

Algorithm SRRI

Input: an arbitrary history h
Output: a scheduled history h^* with the minimum number of misses

- Given h, suppose that we use replacement policy $\mathcal{R}$.
  1. If there exists $\sigma_{\mathcal{R}}(a)$ for any request a, shift $\sigma_{\mathcal{R}}(a)$ to immediately before a.
  2. Iterate step 1 from beginning to end of h.

The following theorem proves the correctness of the algorithm.

Theorem 2.1 *Given an arbitrary history h of length n, let H be a set of histories containing the same requests as h. The algorithm* **SRRI** *produces a history h^* with the minimum number of misses in H. That is, h^* has $O(1)$ misses for h having $O(n)$ misses.*

Proof The algorithm **SRRI** produces a history h^* such that no two misses for a request occur in h^*. Since every page in $U(h)$ causes a miss at least once, h^* has the minimum number of misses, $|U(h)|$, in H. ∎

There are several criteria to be achieved, e.g., minimizing reordering cost, maximum delay, mean delay, and so forth. We have a few choices of page replacement policy by which criterion should be achieved. Now we consider two policies in the following. The number of reordering between $p^{(i)}$ and $q^{(j)}$ $(i < j)$ is defined as the distance from $p^{(i)}$ to $q^{(j)}$. We also call it *an exchange* (of $p^{(i)}$ and $q^{(j)}$). The *reordering cost* for q, denoted by $rc(q)$, is the sum of the necessary number of reordering for all the references of q. Let us also define *starvation* as some requests more than distance D are possibly reordered.

Theorem 2.2 *If we adopt LSD as the page replacement policy $\mathcal{R}$,* **SRRI** *can minimize the reordering cost and is free from starvation.*

Proof Suppose that the request for $a^{(i)} \in I_0$ causes a miss and the page $b^{(j)}$ is picked by *LSD*. After exchanging $a^{(i)}$ and $b^{(j)}$, their positions in h are changed to $b^{(i)}$ and $a^{(i+1)}$. Thus if we sum up all the distances over h, the reordering cost for b,

$$rc(b) = \sum_{k \geq 0, b^{(j_k)} \in U(h)} d(a^{(i+k)}, b^{(j_k)}),$$

is obtained.

Since $|U(h)| - m$ requests in I_0 cause misses and *LSD* picks a page for each miss, we evict $|U(h)| - m$ pages whose reordering costs are not larger than those of other m pages. Next, let us consider the page x with largest $rc(x)$ among the pages to be evicted (i.e., within $|U(h)| - m$ pages). Without loss of generality, we assume that $rc(x) < n$, that is, the length of history is larger than $rc(x)$. Any request for y, appearing after the distance of $rc(x)$ from the current request, is not reordered by **SRRI** because of $rc(x) < rc(y)$. If we regard such $rc(x)$ as D, *LSD* provides no starvation. ∎

Let $\mu(x)$ be the number of references of x after I_0. If A is a set, let $\mu(A)$ be the set of the number of references for each element in A.

Theorem 2.3 *If we adopt LFU as the page replacement policy $\mathcal{R}$,* **SRRI** *can minimize the maximum delay.*

Proof Suppose that the requests for $r_1, r_2 \ldots, r_t$ $(t = |U(h)| - m)$ cause misses in I_0, and that for each miss the page $x_1, x_2 \ldots, x_t$ are picked, respectively. Then we can express $x_1 = \min\{\mu(x) | x \in C_{LFU}(r_1)\}$, $x_2 = \min\{\mu(x) | x \in C_{LFU}(r_2)\}$, ..., and $x_t = \min\{\mu(x) | x \in C_{LFU}(r_t)\}$. Combining $(t-1)$st and tth expressions, we have $\mu(x_{t-1}) < \mu(x_t) < \min \mu(C_{LFU}(r_t) - x_t)$. Combining $(t-2)$nd and $(t-1)$st expressions provides $\mu(x_{t-2}) < \mu(x_{t-1}) < \mu(x_t) < \min \mu(C_{LFU}(r_t) - x_t)$, and so forth. Hence *LFU* picks the t pages whose usage is less frequent than others.

The maximumly delayed requests are within $[u, v]$, where u is the last request in I_0 and v is the request immediately before the earliest $y^{(j)}$ for all y and j. These requests are delayed once for an exchange. Since *LFU* picks t pages causing the minimum number of exchanges, **SRRI** minimizes the maximum delay. ∎

The following theorem describes the efficiency of **SRRI**.

Theorem 2.4 *The algorithm* **SRRI** *runs in $O(n)$ time.*

Proof Suppose that **SRRI** keeps an array corresponding $U(h)$ and that it counts the number of references of each page by scanning a history once. Next, it determines cache contents for miss requests in I_0, then it scans the history once more and shifts requests. ∎

3 Semi-Online Scheduling

Now we consider the semi-online version of **SRRI**, where we have an input queue keeping some future requests. Though the term "online" is usually used when we cannot use future information, our method assume that we can know it to some extent. Hence, we do not call it "online" but "semi-online".

Suppose that a page q is now replaced by a miss request r, and the end of queue at the time is end_r. We say that the output history is **queue interval safe** if the evicted page q will not be called back during the interval $[r, end_r]$ for any r and end_r.

3.1 Algorithm

Here we present a semi-online version of **SRRI**, called **partial LRU + SRRI**, that is, a hybrid method of **LRU** and **SRRI**. Let $AA(r)$ be a set of pages in $C_{\mathcal{R}}(r)$ ($AA(r) \subseteq C_{\mathcal{R}}(r)$) for which requests have already arrived in the queue at the current request r. If $|AA(r)| \geq k$ for some constant k ($1 < k < m$), the replacement policy $\mathcal{R}$ specifies a page s to be evicted. Then the future requests for s (in the queue) are shifted to its residence interval. If $|AA(r)| < k$, a page is evicted by the past information. That is, the least recently used page in the past is specified from the unappeared requests in the queue. Note that once evicted page is determined by $\mathcal{R}$, it will not be changed [2] for simplicity.

Algorithm partial LRU + SRRI

Input: arbitrary page requests
Output: a scheduled history h^* with queue interval safe.

- For an input request q, iterate 1 and 2 for each miss request p until the latest $AA(p)$ is defined.
  1. Add q to $AA(p)$ if $|AA(p)| < k$ and $q \in C_{\mathcal{R}}(p) - AA(p)$.
     Pick $s \in AA(p)$ by the replacement policy $\mathcal{R}$ if $|AA(p)| \geq k$.
  2. Shift s to immediately before p for every s in the queue satisfying $\sigma_{\mathcal{R}}(p) = s$. (**SRRI**)
- For the current miss request r, examine if $|AA(r)| \geq k$.
  - If so, the page s picked by $\mathcal{R}$ is replaced by r.
  - If not, the page with least recently used (in the past) in $C_{\mathcal{R}}(r) - AA(r)$ is replaced by r. (**partial LRU**)

The following theorem proves correctness.

Theorem 3.1 *Any history h^* produced by* **partial LRU + SRRI** *is queue interval safe.*

Proof Let s be the page replaced by any miss request p, and end_p the end of queue at the time. Assume that the request of s is contained in $[p, end_p]$. Then we have to execute the step 2. As a result, s is shfted to immediately before p. This property is kept until p is executed. Thus the output history h^* is queue interval safe. ∎

For the part of the partial *LRU*, it takes constant time if we keep the information of $AA(r)$ for each miss request r. The efficiency of **partial LRU + SRRI** is shown in the following theorem.

[2] Some picked request may not be picked by $\mathcal{R}$ rule if new request comes.

Theorem 3.2 *Let l be the queue length at any time, then it takes $O(l)$ to shift all requests for an input request q.*

Proof Suppose that the top request p of the queue has the latest $AA(p)$ with $k-1$ elements. If the input request q increases $|AA(p)|$ to k and picks a page by $\mathcal{R}$ rule, the algorithm computes next miss request and its AA value (step 1). Then it shifts the picked page (step 2). These steps can be continued until the end of queue. Thus it takes $O(l)$. ∎

The following theorem investigates the appropriate value of k.

Theorem 3.3 *If we intend to achieve both high performance and low reordering cost, we should set k to the value minimizing the weighted sum (α, β: weight)*

$$B(k) = \alpha \cdot \frac{m+1}{k(k+1)} + \beta \cdot q(H_m - H_{m-k}),$$

where m is the cache size and q is a constant satisfying $m < q < |U(h)|$. Note that H_n is the nth harmonic number.

Proof To achieve low reordering cost, we have only to contain the page with *LSD* property in the cache as probably as possible. This can be modelled as follows. Suppose that m balls numbering from 1 to m are in the box, where ith number means the page with ith *LSD* property. If we pick k balls at random, find the expected smallest number. The probability that r is the smallest number is $(1/m)\binom{m-r}{k-1}/\binom{m-1}{k-1}$ because arbitrary number more than r is only allowed after we pick r. Thus we have the expected smallest value

$$\sum_{r=1}^{m-k+1} r \frac{1}{m} \frac{\binom{m-r}{k-1}}{\binom{m-1}{k-1}} = \frac{m+1}{k(k+1)}.$$

To achieve high performance, we have only to shift as many miss requests as possible. Recall that the algorithm does not shift them until k requests in the cache appear in the queue. Hence if the pages are assumed to be requested at random, this can be modelled as follows. Suppose that we have a complete graph with q nodes ($m < q < |U(h)|$), where each node and edge represent a page and a transition, respectively. Find the expected number of requests until k nodes appear. Let Y_j be the number of requests from $j-1$ to j pages appear in the queue, and p_j its probability. Since Y_j has a geometric distribution with $Pr\{Y_j = t\} = (1-p_j)^{t-1}p_j$, where $p_j = \{m-(j-1)\}/q$, we have $E[Y_j] = 1/p_j = q/(m-j+1)$. Thus

$$\sum_{j=1}^{k} E[Y_j] = q(H_m - H_{m-k}).$$

Since both are convex, minimizing the weighted sum, $B(k)$, derives a Pareto optimal solution. [2] ∎

3.2 Simulation

To evaluate our **SRRI**, we execute simulation experiments and compare with other two methods. One is the well-known *LRU*, and the other is a lookahead method with partial *LRU* (without **SRRI** operations). We adopt *LSD* as the page replacement policy $\mathcal{R}$.

Our simulation model is composed of a slow memory, a cache, an input queue, and a semi-online sequence of requests. The inter-arrival time of them is simulated as exponential distribution with mean *lambda*. They are accumulated in the queue. The top of the queue, the current request r, is picked and examined if it is a hit or a miss. If it is a hit, we just increase the current time by *hitcost*. Otherwise, we have to determine the page to be evicted.

Constant	Value	Meaning
$U(h)$	100	Number of distinct pages in h
maxreq	3000	Number of executed requests
k	5	$\mathcal{R}$ picks from k arrived requests
hitcost	0.001	Execution time for a hit
misscost	1	Execution time for a miss

Table 1: Constants

Parameter	Range	Standard	Meaning
lambda	1—10	5	Mean inter-arrival time of requests
csize	5—50	10	Cache size

Table 2: Parameters

We execute two experiments. Experiment 1 examines the effect of frequency of requests, *lambda*, varying from 1 to 10 (keeping $csize = 10$). Experiment 2 examines the effect of a cache size, *csize*, varying from 5 to 50 (keeping $lambda = 5$). Note that the number of executed requests is 3000 for a trial, and the number of misses is averaged over 5 trials. The constants and parameters are summarized in Tables 1 and 2.

The results of Experiment 1 are depicted in Figure 1(a). It is very interesting that our method shows a high performance even if requests are in high contention. This is because a long queue is available. The results of Experiment 2 are depicted in Figure 1(b). When the cache size is small, our method outperforms others. As the cache size grows, **partial LRU** gets better.

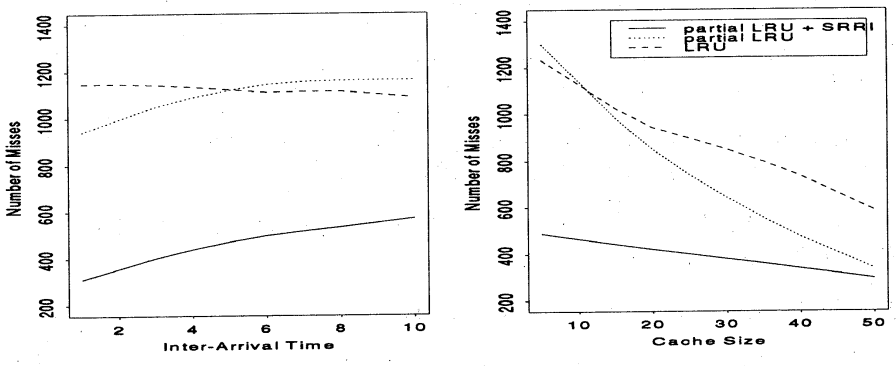

Figure 1: (a) Varying mean inter-arrival time (b) Varying cache size

4 Conclusion

We investigated an effect of lookahead scheduling requests on the paging problem. Given an off-line history, we showed that scheduling requests can reduce the number of misses. Additionally, the procedure is efficient. Next, we considered the case in which we can know the future requests to some extent, called semi-online. We developed an efficient algorithm for the semi-online case, and showed that it has a property of queue interval safe. The algorithm uses a parameter k from which $\mathcal{R}$ picks

a page. We investigated what k value is appropriate for high performance and low reordering cost. At last, we executed simulation experiments for semi-online sequences of requests, assuming that the queue length is determined by frequency of requests and access costs. As a result, our method **partial LRU + SRRI** turned out to be far superior to **partial LRU** and **LRU**. In particular, our method showed a high performance when high frequency of requests.

Acknowledgement

We would like to express sincere appreciation to Professor T.Hasegawa, Professor T.Ibarki and Professor N.Katoh of Kyoto University, and Professor S.Nishio of Osaka University for their constant encouragement. In particular, we got a useful information on Theorem 3.3 from Prof. N.Katoh.

References

[1] Albers,S., "On the Influence of Lookahead in Competitive Paging Algorithms", *Algorithmica, 18* (3) , pp.283–305, 1997.

[2] 馬場則夫, 坂和正敏, "数理計画法入門", 共立出版, 1989.

[3] Belady,L.A., "A Study of Replacement Algorithms for a Virtual-Storage Computer", *IBM Syst. J. 5* (2), pp.78–101, 1966.

[4] Ben-David,S. and Borodin,A., "A New Measure for the Study of On-line Algorithms", *Algorithmica 11*, pp.73–91, 1994.

[5] Breslauer,D., "On Competitive On-line Paging with Lookahead", *Proc. 13rd Symp. Theoretical Aspects of Computer Science*, pp.593–603, 1995.

[6] Coffman,E.G., Klimko,L.A. and Ryan,B., "Analysis of Scanning Policies for Reducing Disk Seek Times", *SIAM J. Comput. 1* (3), pp.269–279, 1972.

[7] Koutsoupias,E. and Papadimitriou,C.H., "Beyond Competitive Analysis", *Proc. 35th IEEE Symp. Foundations of Coputer Science*, pp.394–400, 1994.

[8] Sleator,D.D. and Tarjan,R.E., "Amortized Efficiency of List Update and Paging Rules", *Comm. ACM 28* (2), pp.202–208, 1985.

[9] Smith,A.J., "Cache Memories", *ACM Comput. Surveys 14* (3), pp.473–530, 1982.

[10] Tanenbaum,A.S., "Modern Operating Systems", *Prentice-Hall*, 1992.

[11] 山家明男, 櫻井幸一, "オンライン先読みページングゲームにおける最適戦略の設計と解析", '97 冬の LA シンポジウム, 1997.